siam 2022 | ACM-SIAM Symposium on Discrete Algorithms

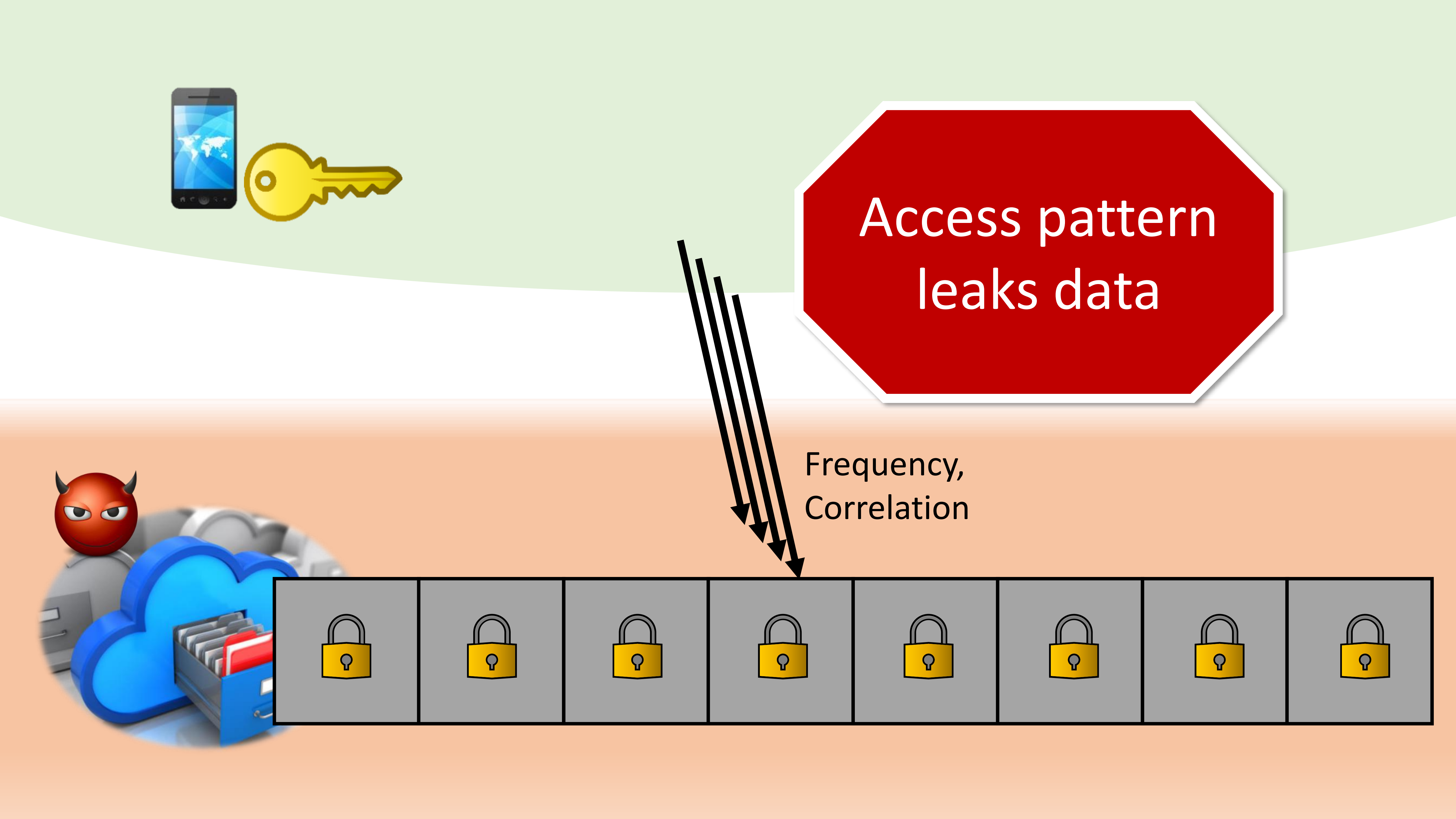
Optimal Oblivious Parallel RAM

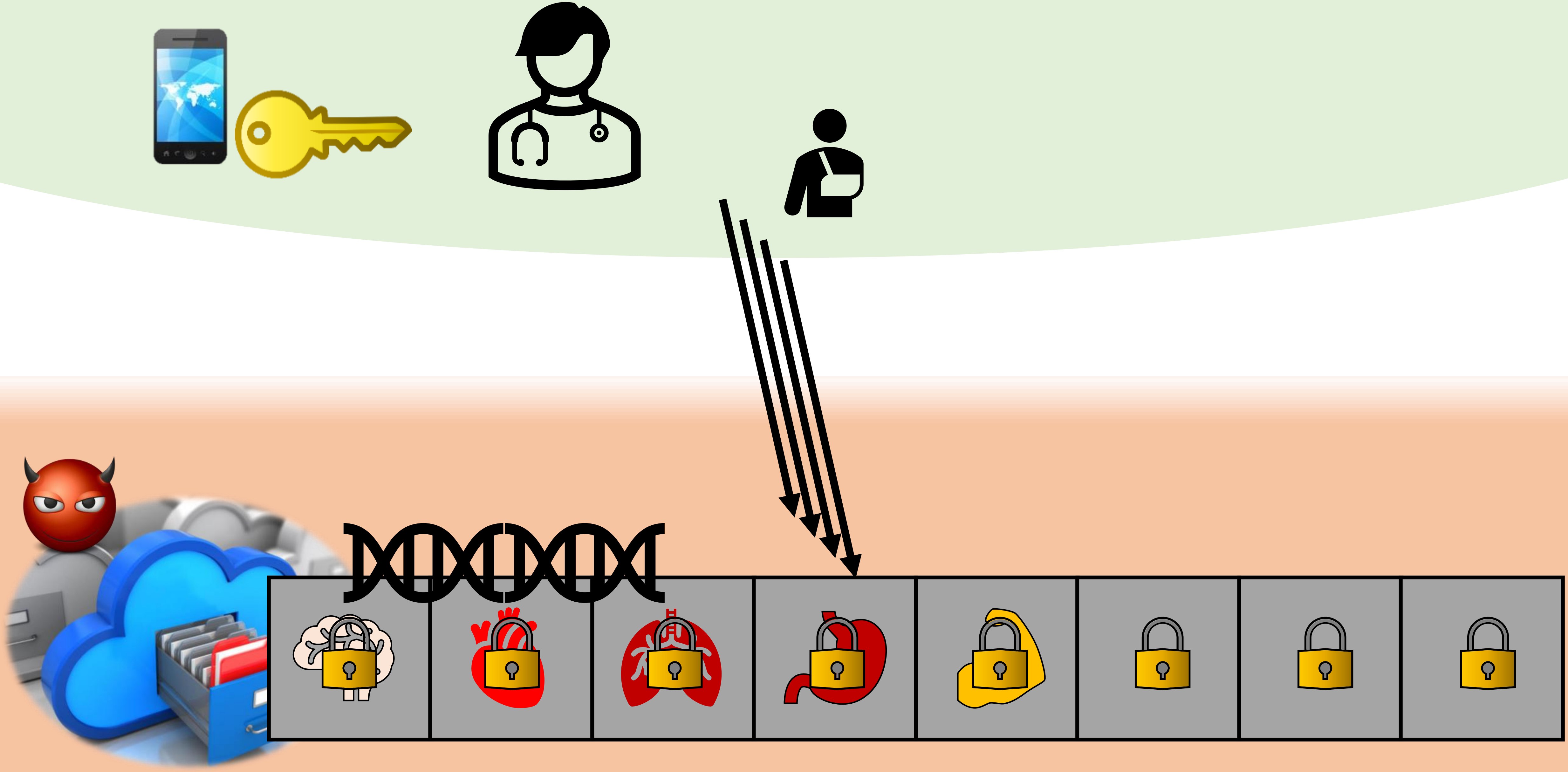
Wei-Kai Lin

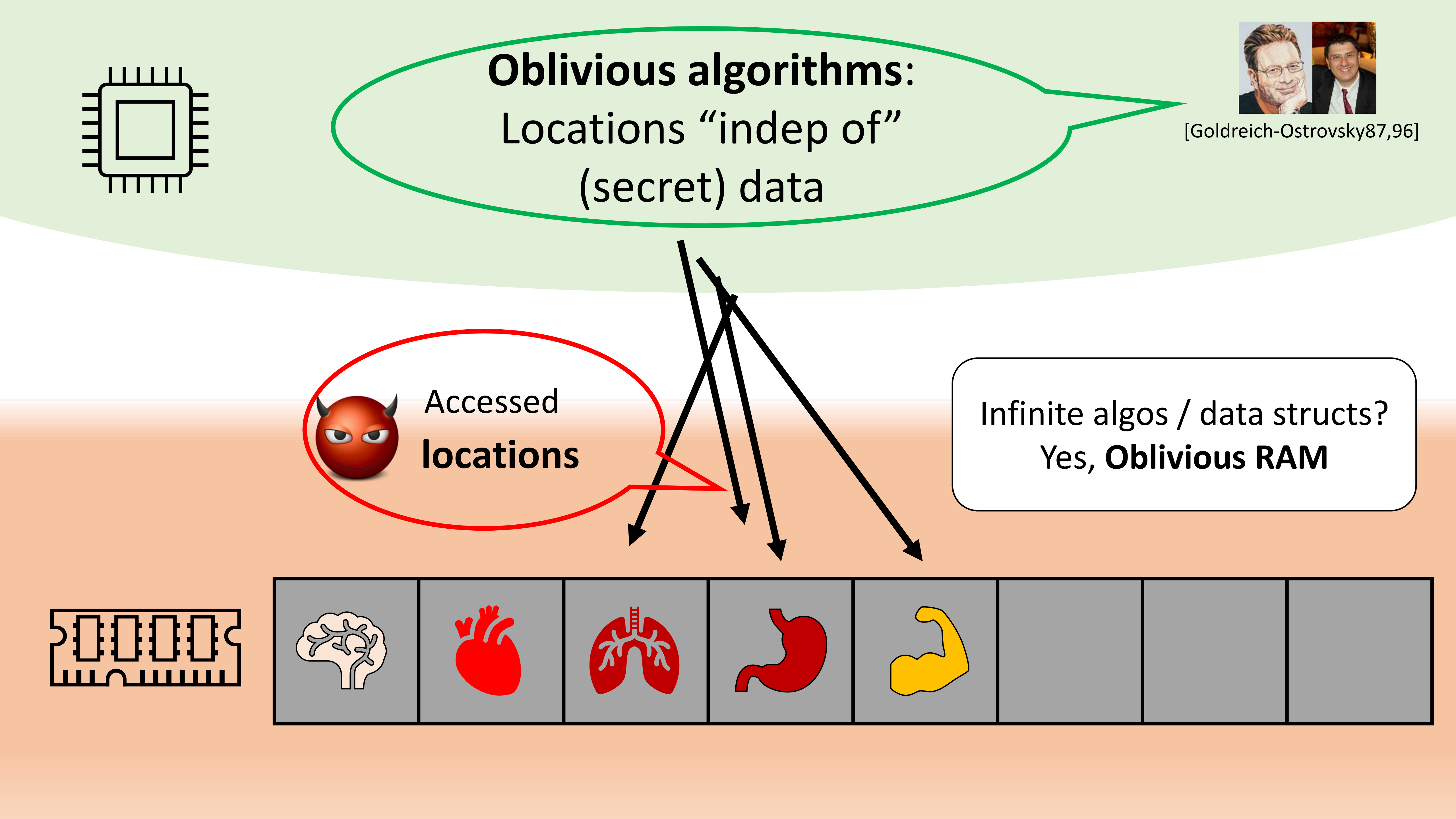
Joint work with

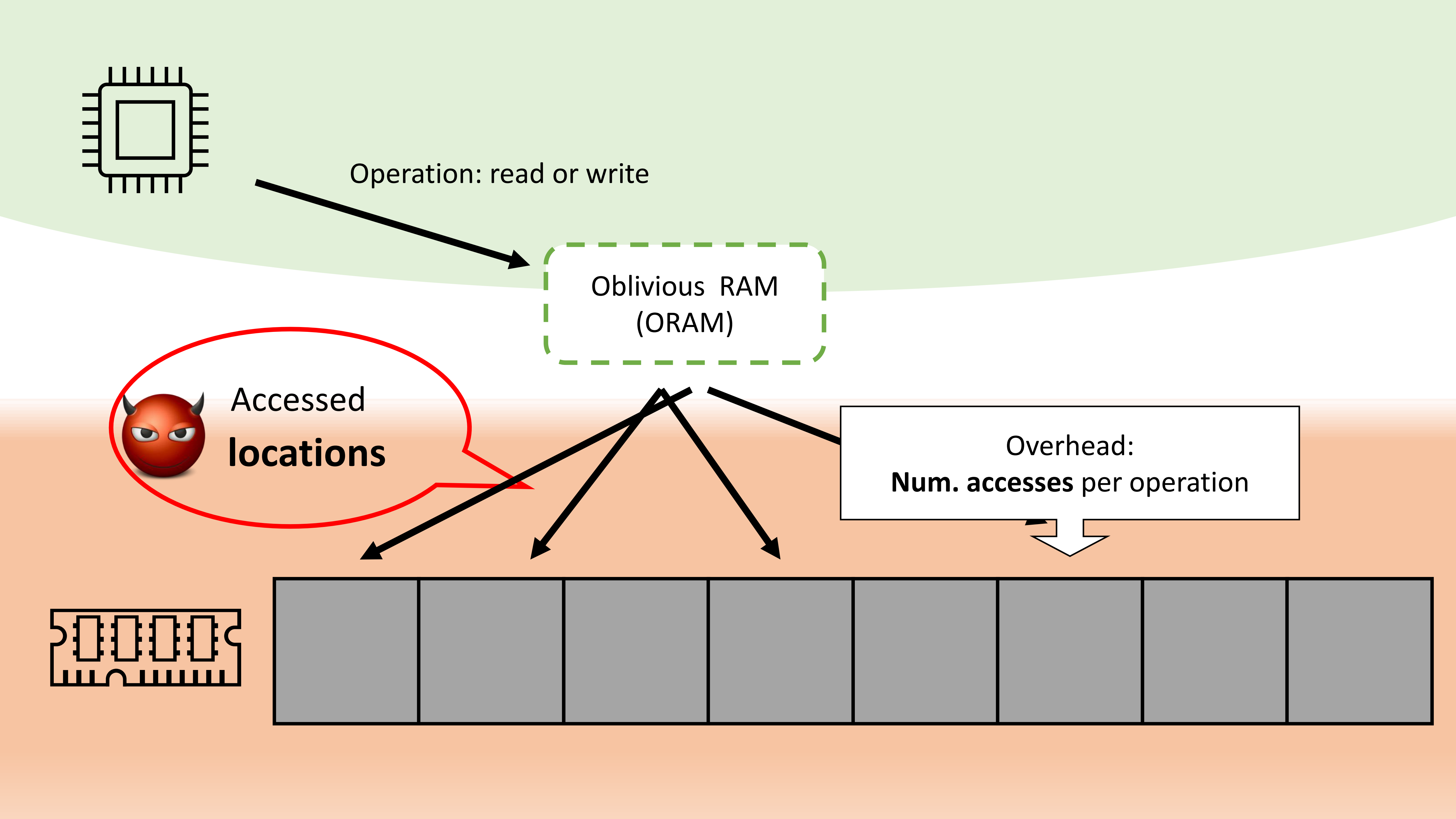
Gilad Asharov, Ilan Komargodski, Enoch Peserico, Elaine Shi

Bar-Ilan University
THE HEBREW UNIVERSITY OF JERUSALEM
NTT Research
Università degli Studi di Padova
Carnegie Mellon University


Access pattern leaks data
Frequency,
Correlation

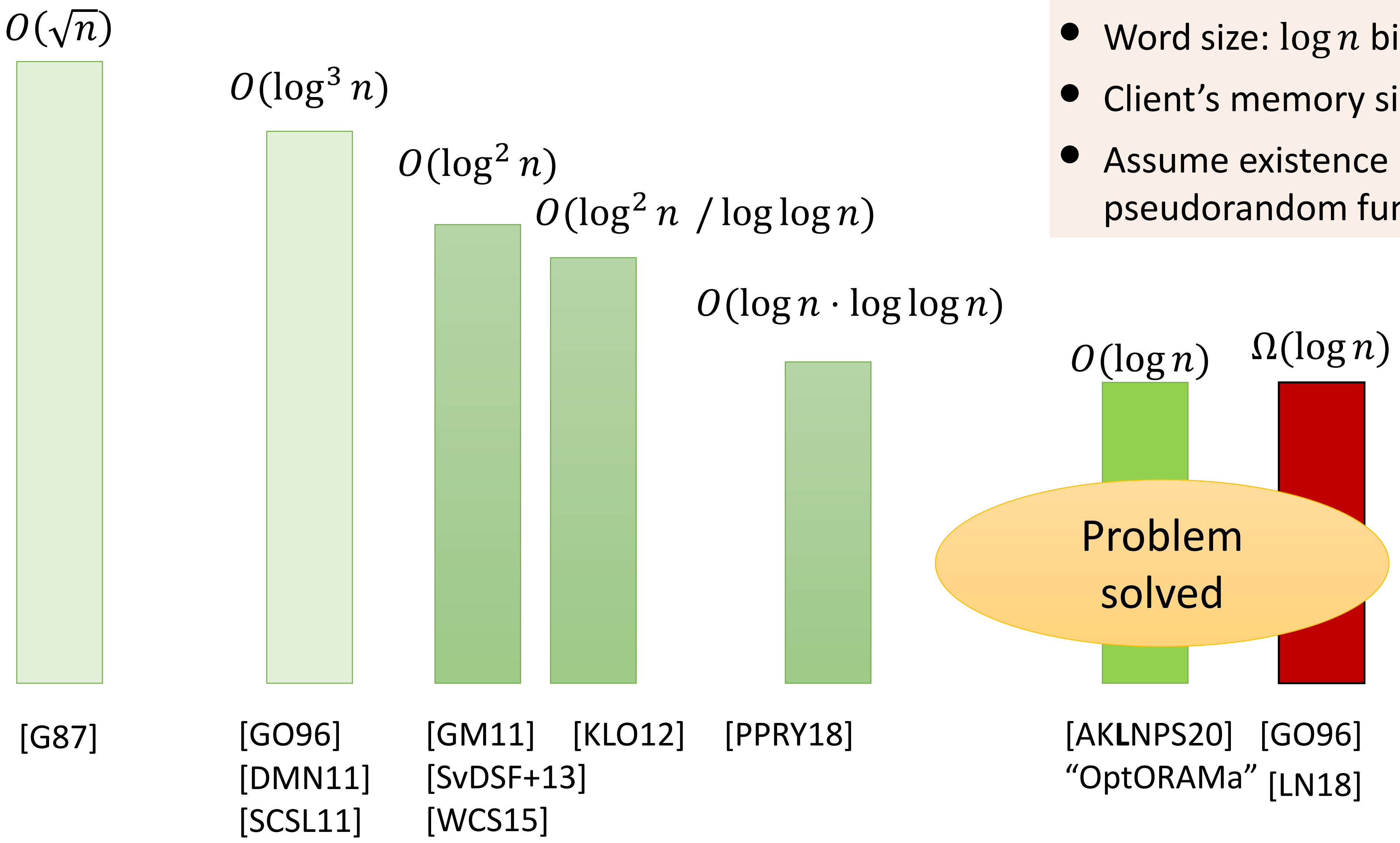


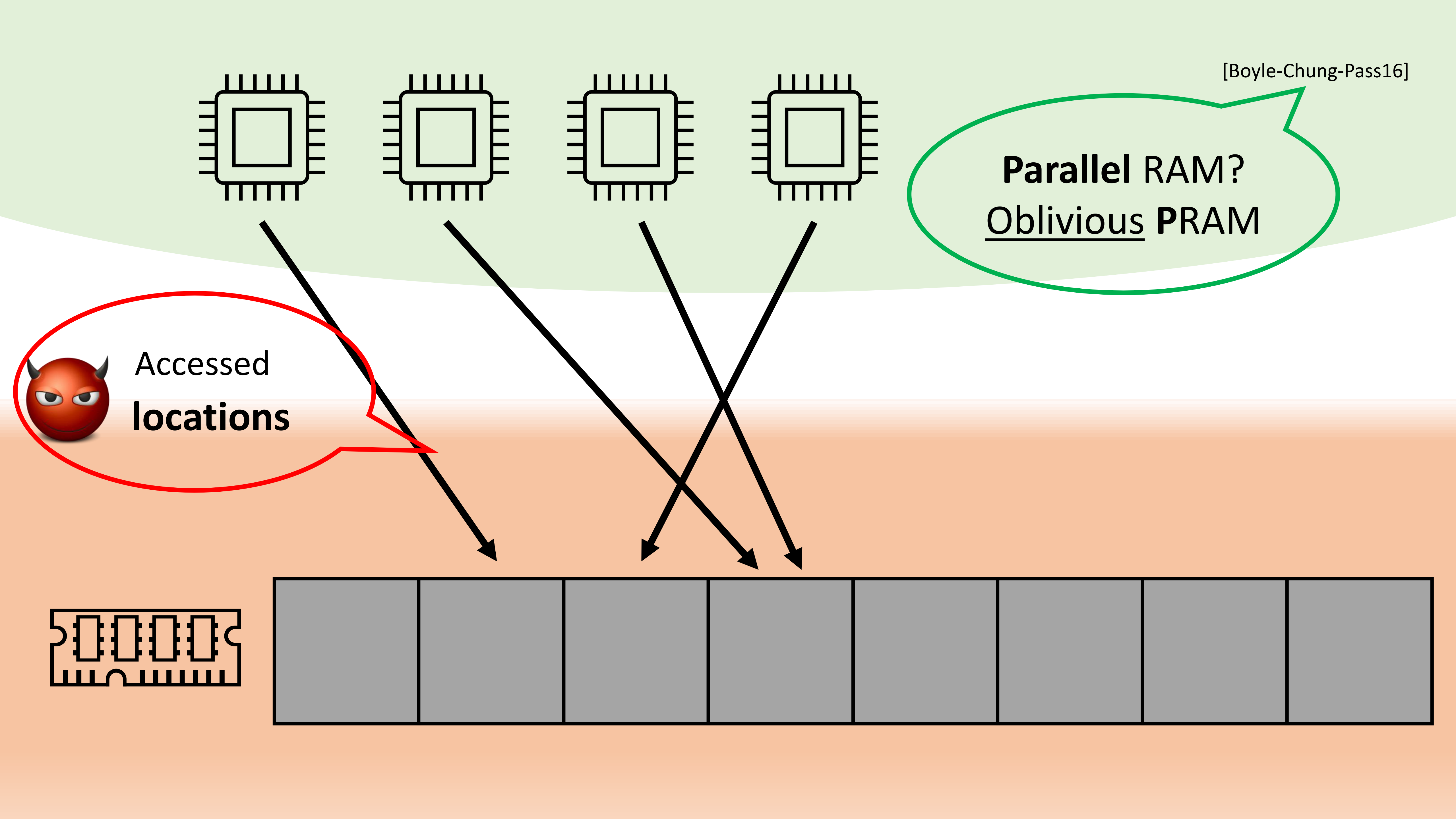

Oblivious algorithms:
Locations "indep of"
(secret) data
[Goldreich-Ostrovsky87,96]
Accessed
locations
Infinite algos / data structs?
Yes, Oblivious RAM

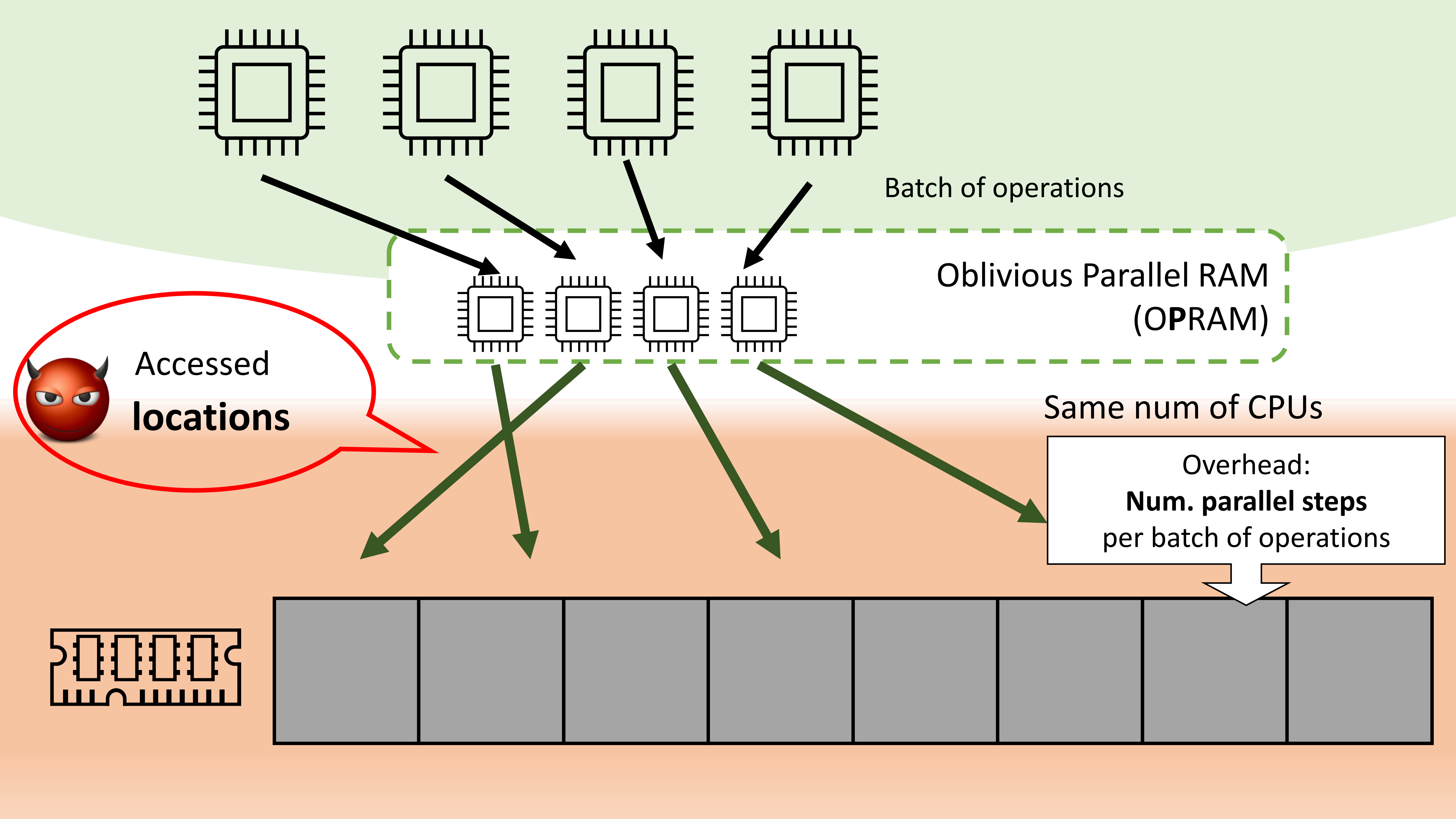

Operation: read or write
Oblivious RAM (ORAM)
Accessed **locations**
Overhead:
Num. accesses per operation

ORAM Overhead, Standard Setting

- n: size of the memory
- Word size: $\log n$ bits
- Client's memory size $O(1)$ words
- Assume existence of cryptographic pseudorandom function

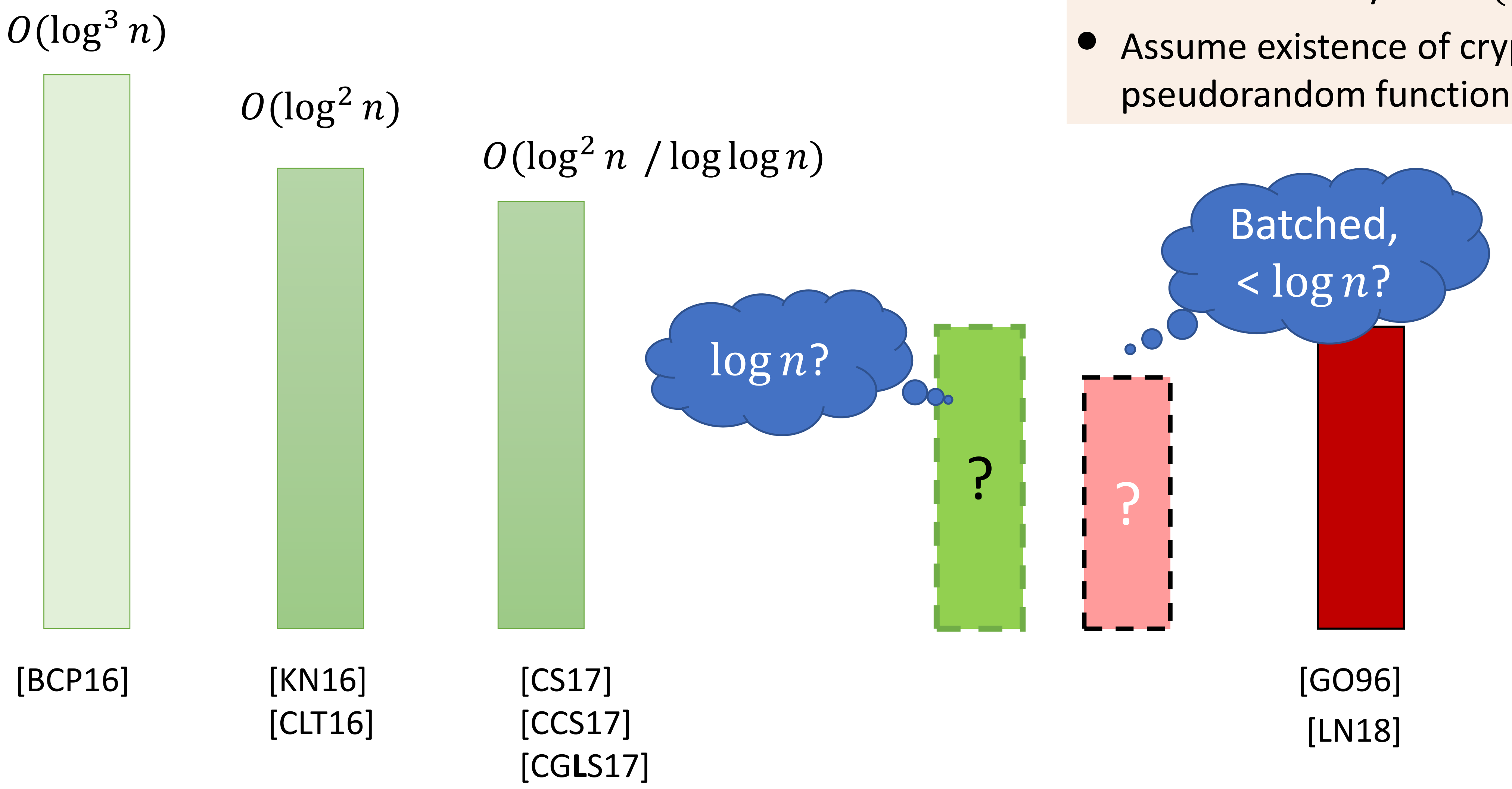



[Boyle-Chung-Pass16]
Parallel RAM?
Oblivious PRAM
Accessed
locations


Batch of operations
Oblivious Parallel RAM
(OPRAM)
Accessed
locations
Same num of CPUs
Overhead:
Num. parallel steps
per batch of operations

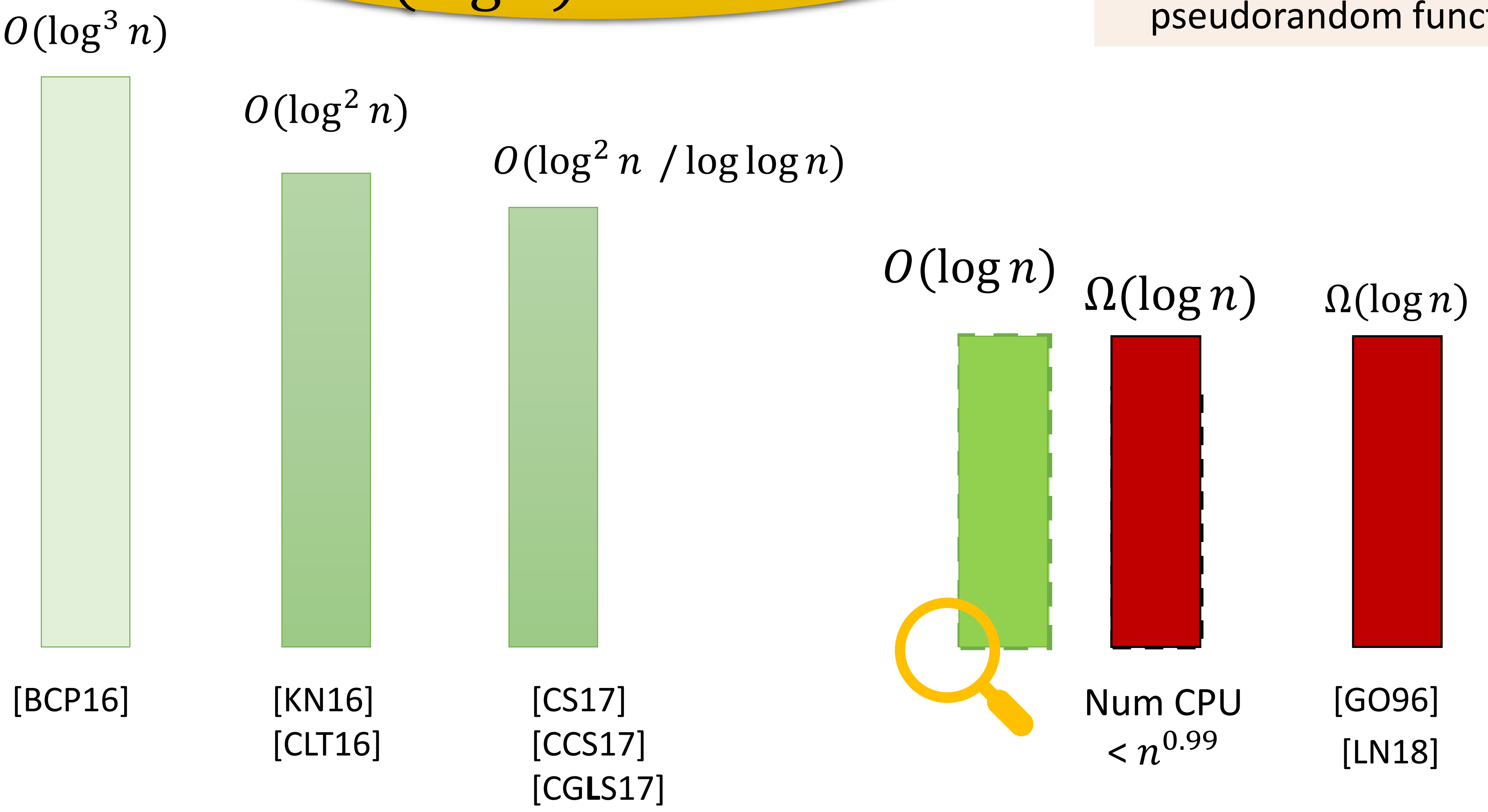
Main question: OPRAM overhead?

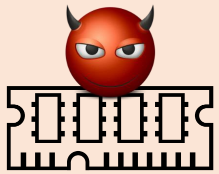
- n: size of the memory
- Word size: $\log n$ bits
- Client's memory size $O(1)$ words
- Assume existence of cryptographic pseudorandom function



Main result:
Optimal OPRAM,
$\Theta(\log n)$ overhead

- n: size of the memory
- Word size: $\log n$ bits
- Client's memory size $O(1)$ words
- Assume existence of cryptographic pseudorandom function



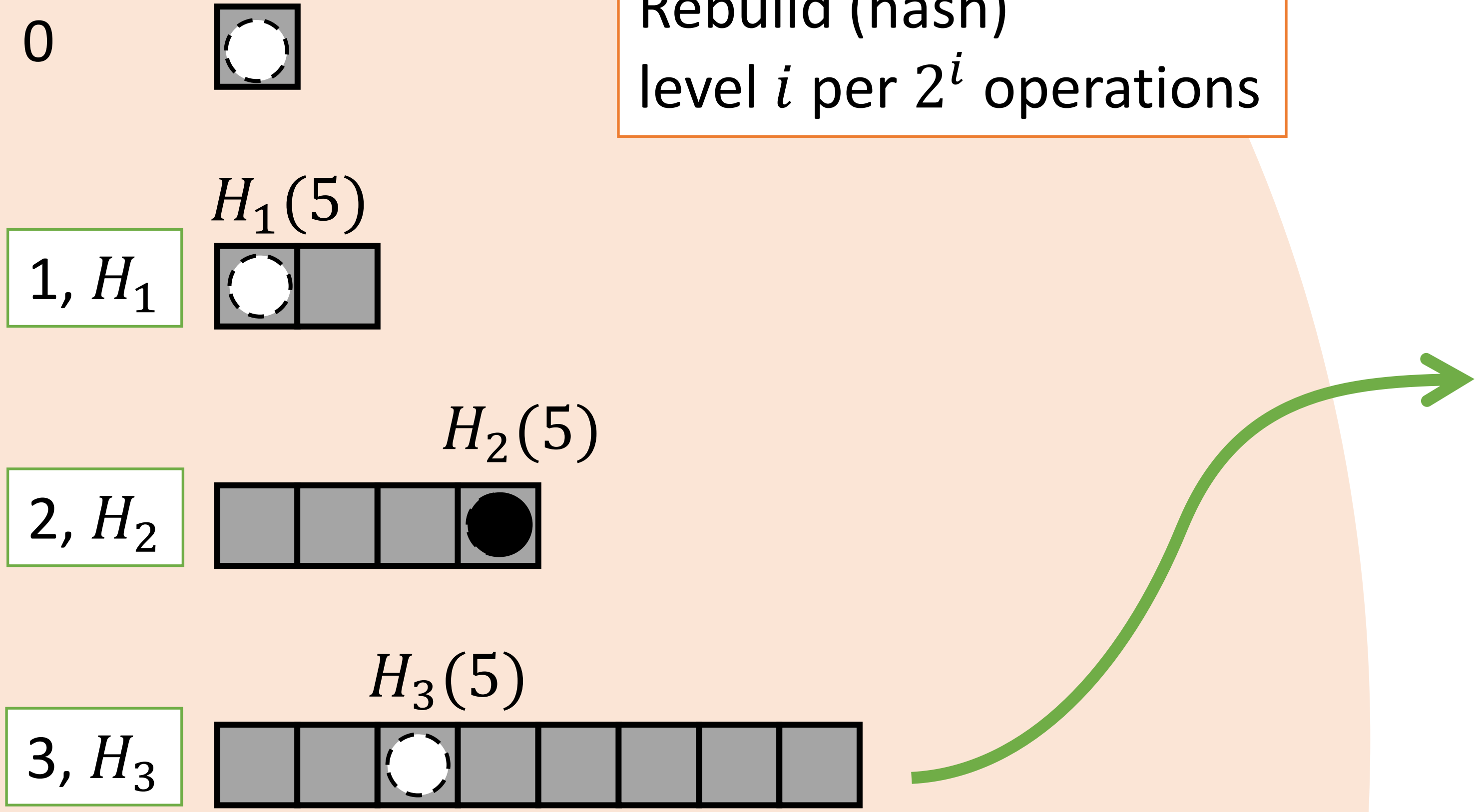


Hierarchical Framework

[Goldreich-Ostrovsky ’87,96]

Level i, capacity 2^i

0



Put ● into Level 0

Rebuild (hash) level i per 2^i operations

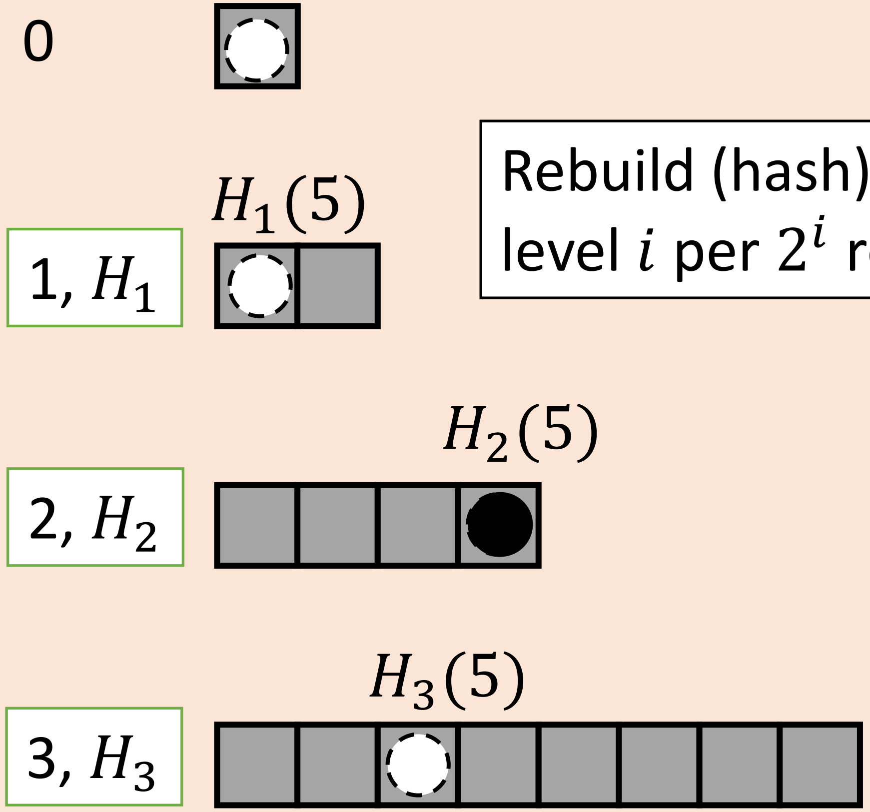
Oblivious Hash Table:

- “Build”: store 2^i balls
- “Lookup” for key k: find and output k and associated ball
- Build + Lookup is <u>oblivious</u>: not reveal balls even adversary sees accesses

Optimal ORAM (sequential)

[PPRY18] "PanORAMa"	[AKLNPS20] "OptORAMa"

Level i, capacity 2^i



Rebuild (hash)
level i per 2^i requests

Oblivious Hash Table

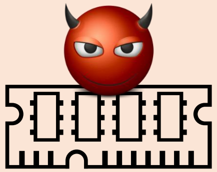
- Build: linear time
- Lookup: const time

Linear time building blocks

"Tight compaction":
Oblivious sorting balls using 1-bit keys

"Intersperse":
Mixing 2 lists uniformly at random

"Toss balls into bins,
and then report loads of bins"



Oblivious Parallel RAM

Also need:
$O(\log n)$ parallel time

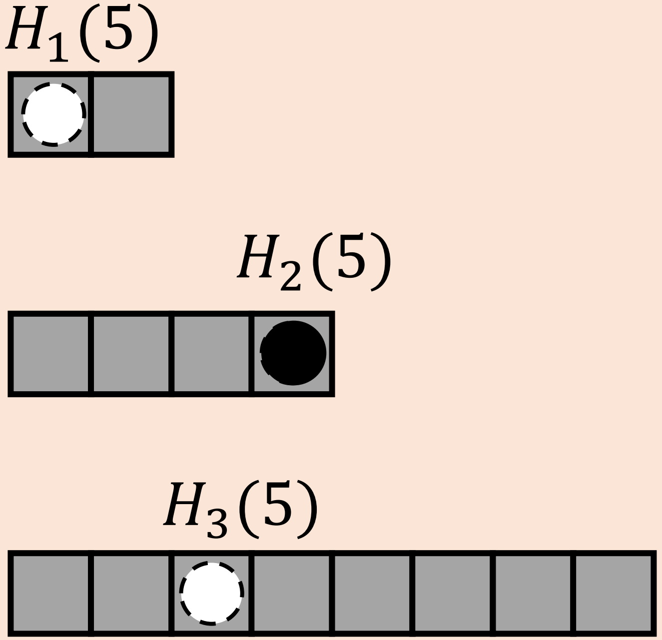
Level i, capacity 2^i

0

$1, H_1$

$2, H_2$

$3, H_3$



Oblivious Hash Table

- Build: linear work
- Lookup: const work

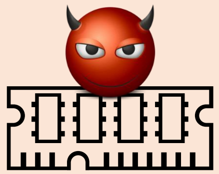
Linear work building blocks

Yes! [AKLPS20]* — "Tight compaction": Oblivious sorting balls using 1-bit keys

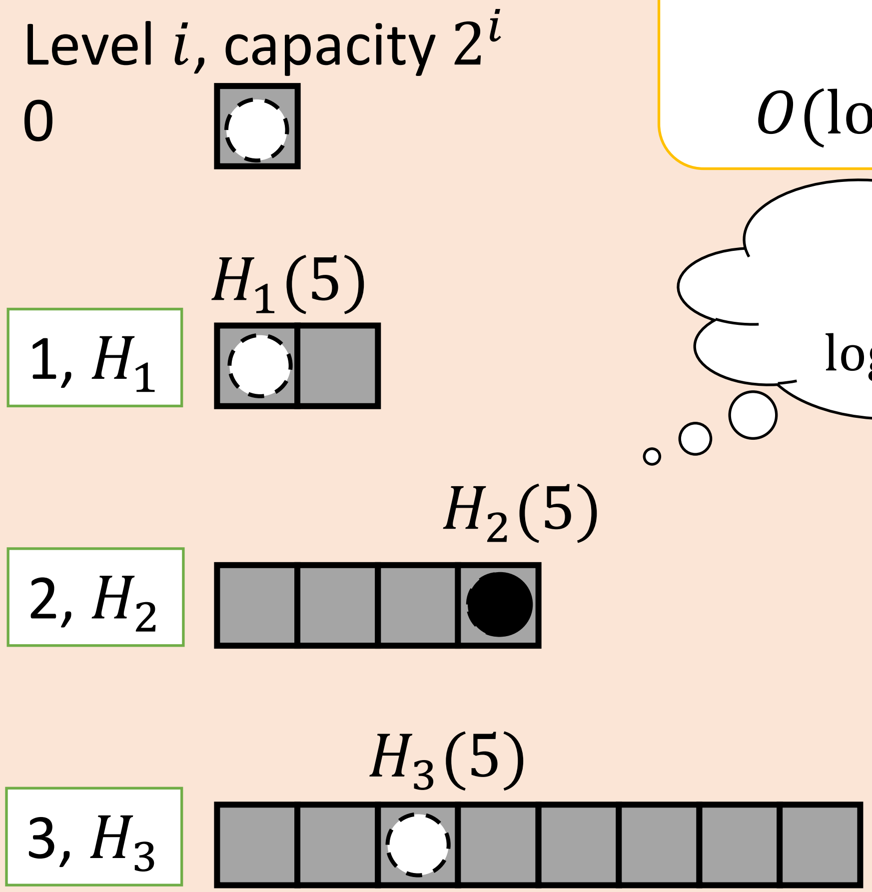
Yes! [This work] — "Intersperse": Mixing 2 lists uniformly at random

Yes! [This work] — "Toss balls into bins, and then report loads of bins"

*Related: "optimal sorting circuits" (Wednesday 4:30, Session 12B)



Oblivious Parallel RAM



Building blocks take:
Linear work
$O(\log n)$ parallel time

$\Rightarrow$ OPRAM: $\sim \log^2 n$ overhead

$\log n$ levels,
each takes
$\log n$ parallel time

Pipeline levels [This work] $\Rightarrow$ OPRAM: $\log n \cdot$ ~~$poly \log\log n$~~ overhead

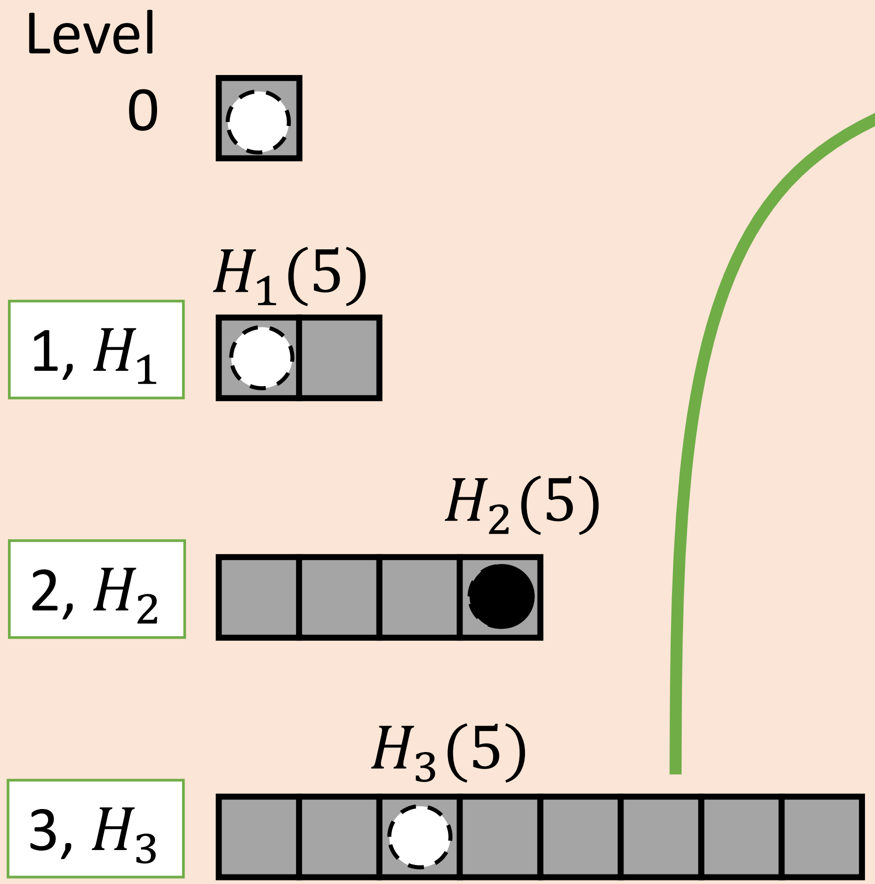
Main Technical Challenge

Oblivious Hash Table

- Build: linear work, $\log n \cdot poly \log \log n$ parallel time
- Lookup: const work

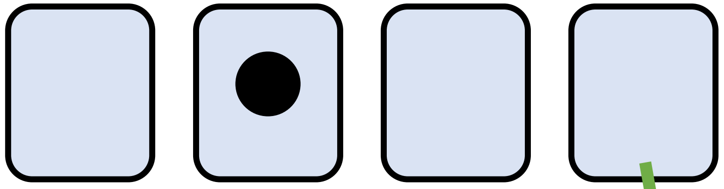
Oblivious Hash Table

[PPRY18] [AKLNPS20]



1. "Balls into bins" hashing

Choose the bin by H(addr)

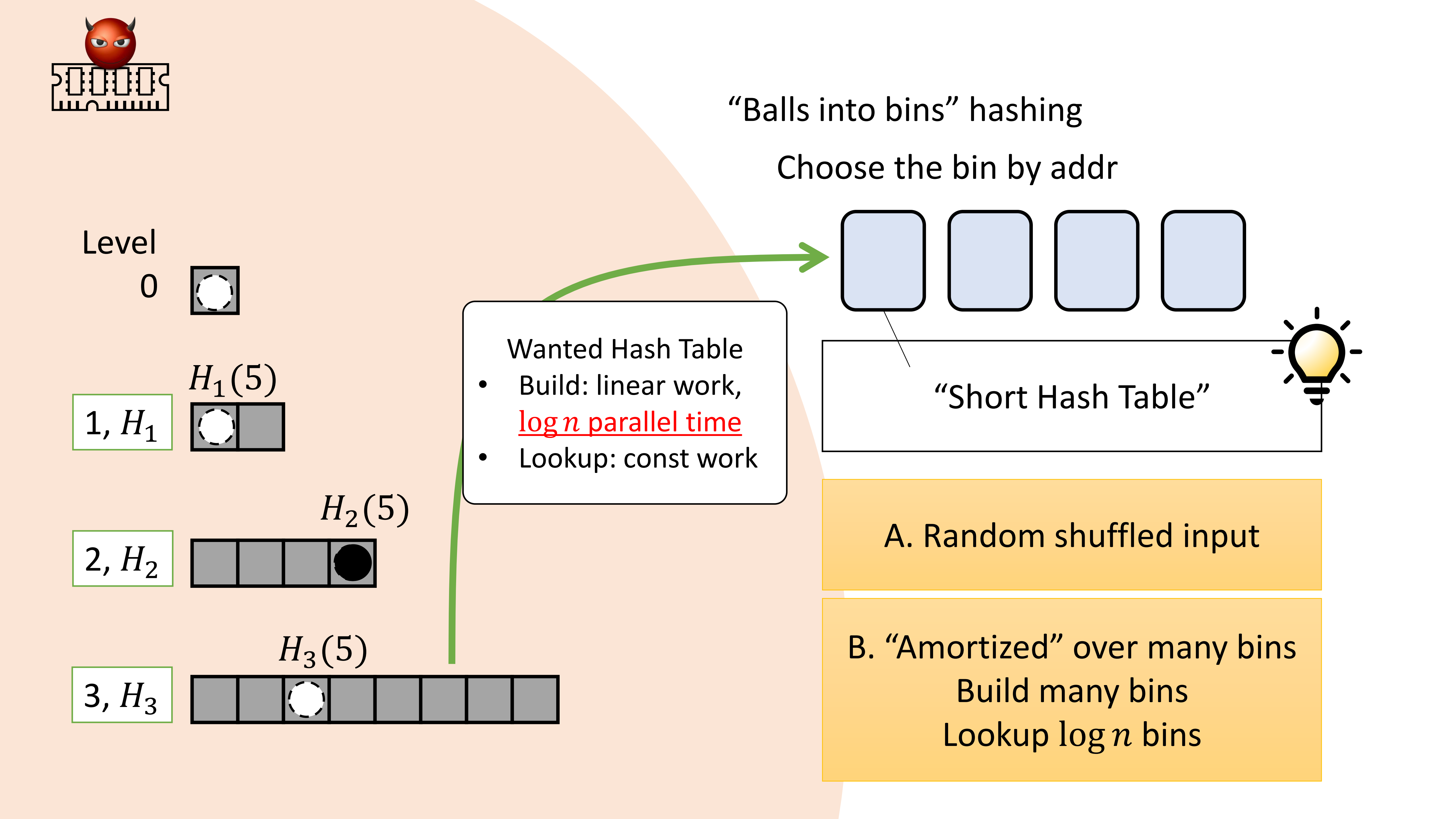


Poly log bin-size ➔ negligible overflow prob.

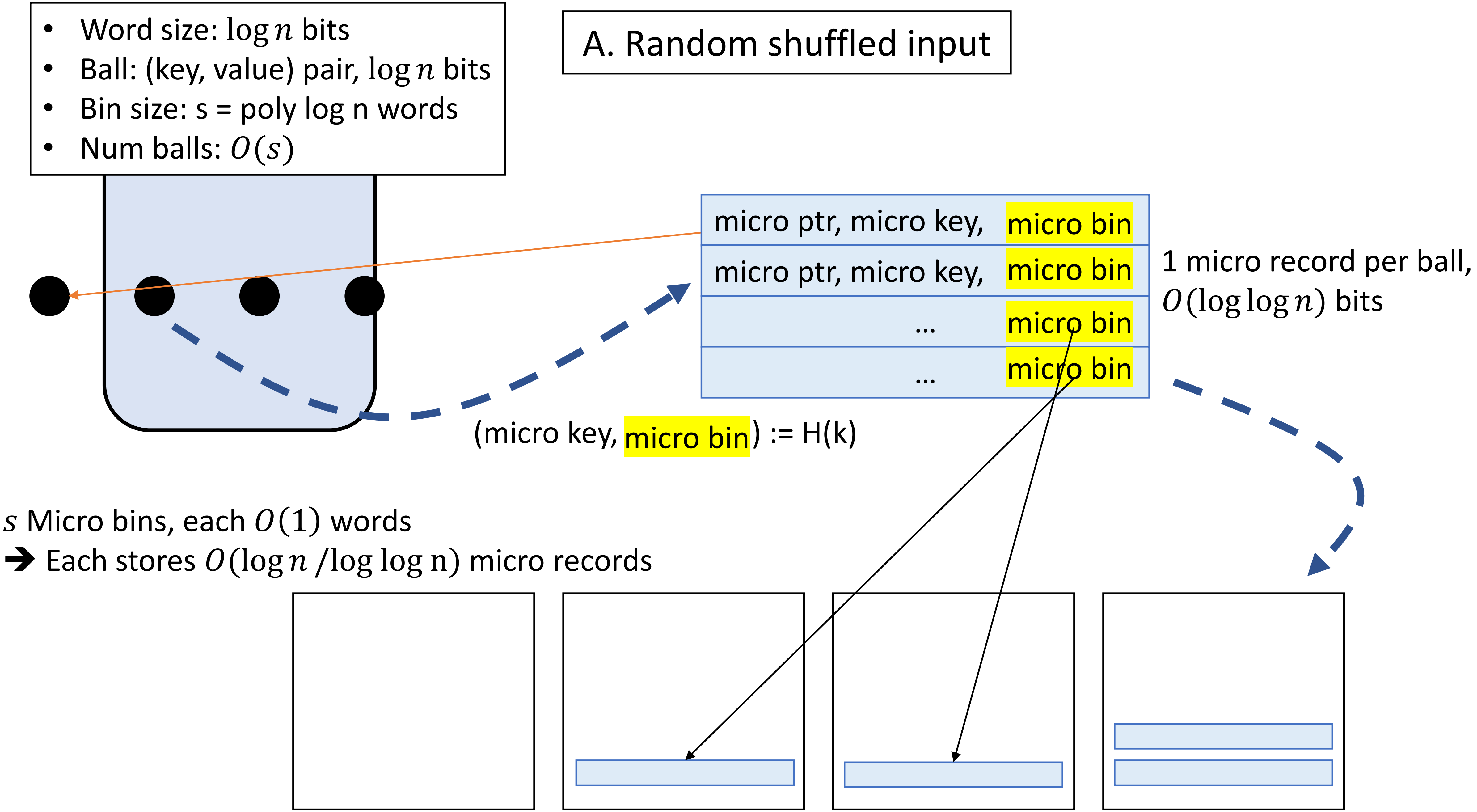
2. "Cuckoo" hashing within each bin

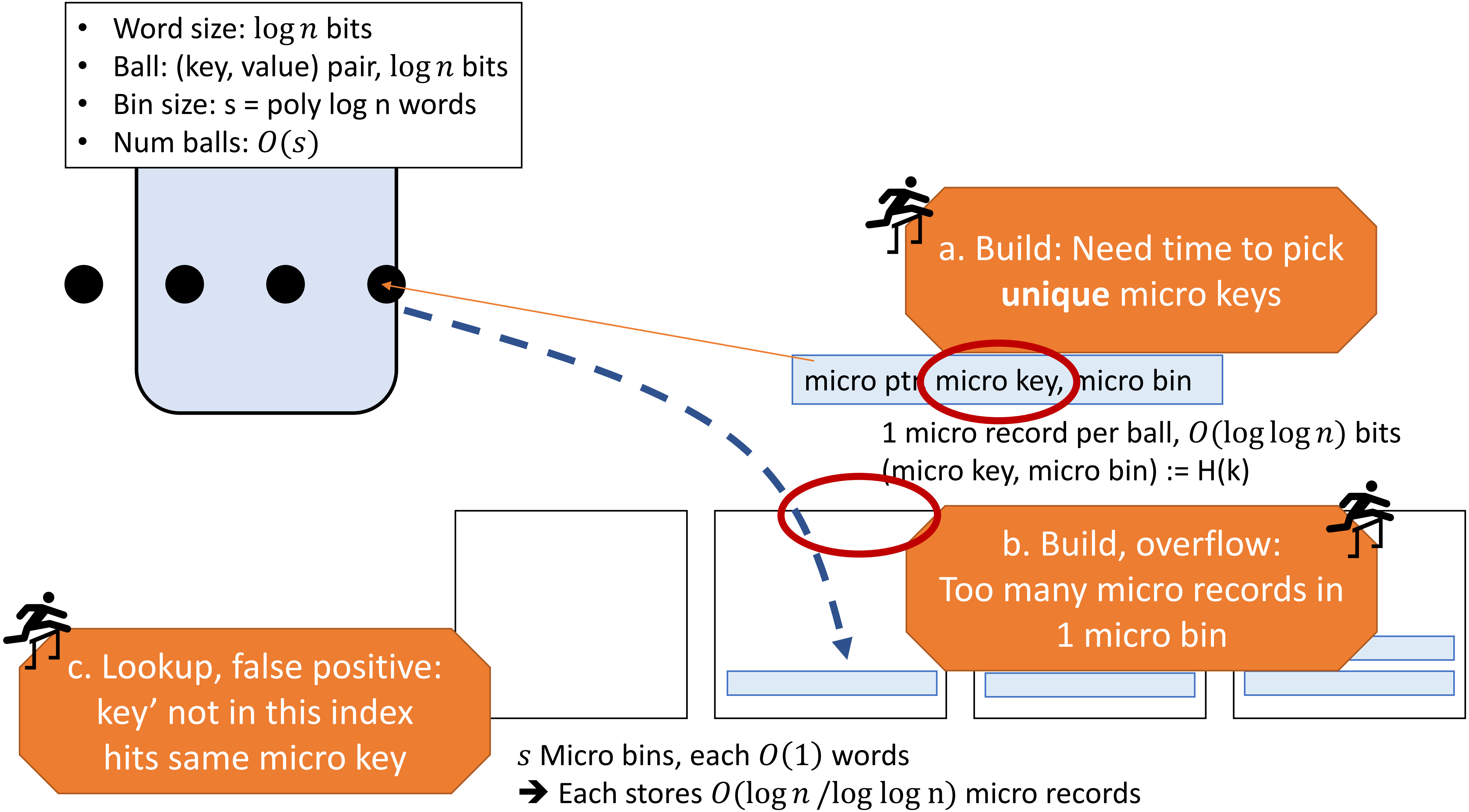
✗ Build cuckoo hashing:
linear work,
<u>$O(\log n \cdot poly \log \log n)$</u>
<u>parallel time</u>

Cuckoo:
Const-time
lookup



Level
0
1, H_1
$H_1(5)$
2, H_2
$H_2(5)$
3, H_3
$H_3(5)$
Wanted Hash Table
• Build: linear work, $\log n$ parallel time
• Lookup: const work
"Balls into bins" hashing
Choose the bin by addr
"Short Hash Table"
A. Random shuffled input
B. "Amortized" over many bins
Build many bins
Lookup $\log n$ bins


• Word size: $\log n$ bits
• Ball: (key, value) pair, $\log n$ bits
• Bin size: s = poly log n words
• Num balls: $O(s)$
A. Random shuffled input
micro ptr, micro key, micro bin
micro ptr, micro key, micro bin
… micro bin
… micro bin
1 micro record per ball, $O(\log \log n)$ bits
(micro key, micro bin) := H(k)
s Micro bins, each $O(1)$ words
➔ Each stores $O(\log n / \log \log n)$ micro records

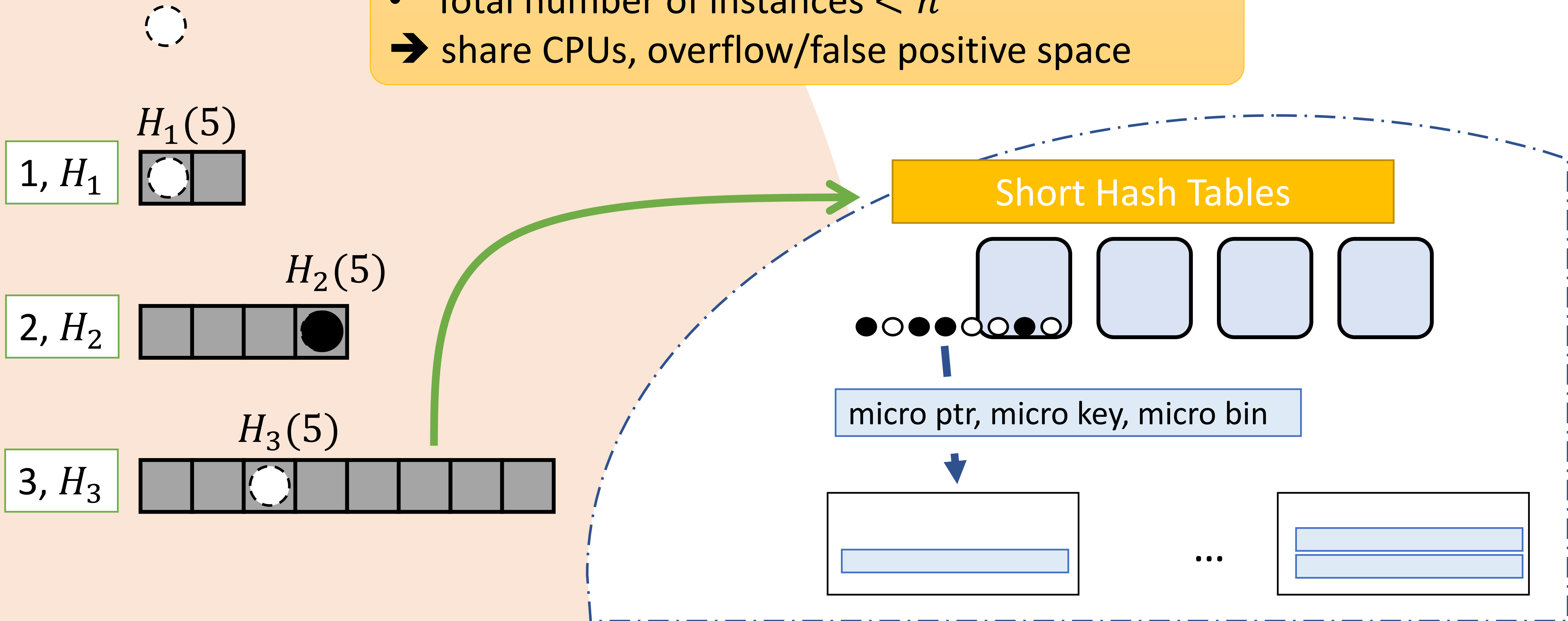


Word size: $\log n$ bits
Ball: (key, value) pair, $\log n$ bits
Bin size: s = poly log n words
Num balls: $O(s)$
a. Build: Need time to pick **unique** micro keys
micro ptr, micro key, micro bin
1 micro record per ball, $O(\log \log n)$ bits
(micro key, micro bin) := H(k)
b. Build, overflow: Too many micro records in 1 micro bin
c. Lookup, false positive: key' not in this index hits same micro key
s Micro bins, each $O(1)$ words
➔ Each stores $O(\log n / \log \log n)$ micro records

B. “Amortized” over many bins

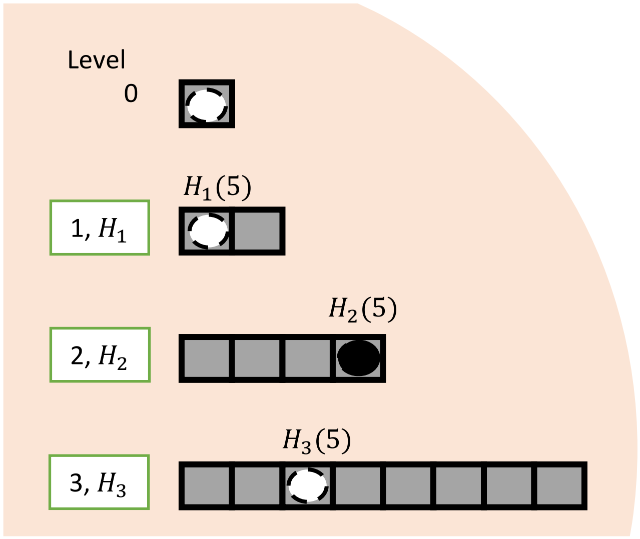
- Build:
  $> poly \log n$ instances of ShortHTs (in parallel)
- Lookup:
  $\log n$ instances of ShortHTs (sequential)
- Total number of instances $< n$

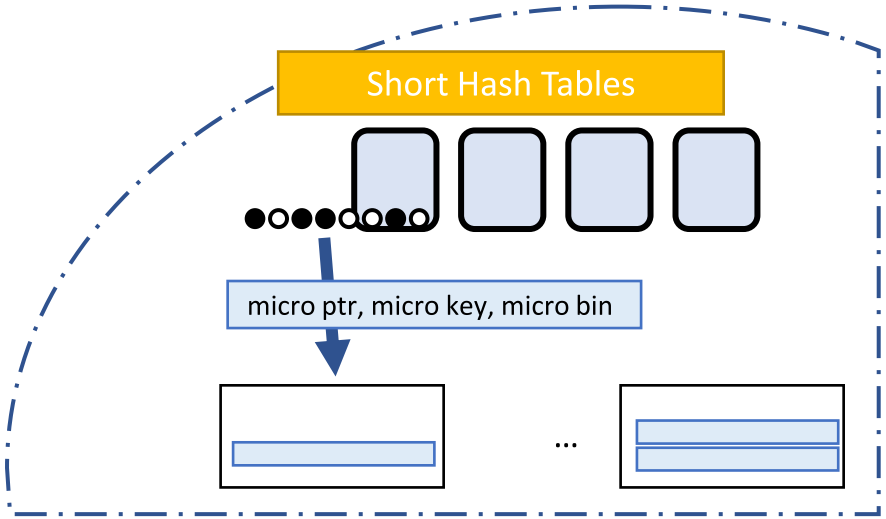
➔ share CPUs, overflow/false positive space



Omitted Details (need oblivious and parallel):

- Pipeline $\log n$ levels of hash tables into Oblivious PRAM
- Building blocks
- Lookup Short Hash Table concurrently





- Extend lower bounds of [GO96] and [LN18] to parallel

Thank you!